# Our GO2 entry

Daniel Bienstock (Columbia), Richard Waltz (Artelys), Jorge Nocedal (Northwestern)

**#2 Overall**

# Outline of talk

- Underlying solver
- Choice of solvers, language(s) and resulting tradeoffs
- Comments on the problem
- Some details on our algorithm
- Future work

# Underlying solver

- **KNITRO**
- General optimizers would call it a "log barrier" solver – an interior point method
- Newton's method under the hood
- Converges to a local minimum of appropriate merit function which balances feasibility and optimality
- **Merit function** (IPOPT: a **filter method**) for stepsize computation
- These are excellent algorithms
- Scale fairly well to large (enough) problems
- Success depends on skill in how we use these methods

# Programming language: why and implications

- We used **Python**
- I code in Python everyday, but I do not like Python. Why not use a modern language, such as **C**?
- I have worked on OPF for some time, but do not fully understand the .RAW format. The transformer specifications are troublesome
- I did not have confidence that I would develop a correct understanding within the scope of the competition
- The scoring methodology was complex and is tied to the data format. In GO1 we replicated some of the scoring code, but GO2 looked more difficult
- We anticipated that our code would need to evaluate multiple candidate solutions for each contingency
- **We decided to incorporate the data reading code and the evaluation code from the evaluation team into our code.**
- GO2 team altered the standard exception handling mechanism in Python. This challenged our debugging.
- Python can be unbearably slow, and Numpy is not always a possibility
- If we had the time we might attempt to redo everything in C. Note: this assumes that the data format and the evaluation code do not change (and they changed late into the competition)

# Modeling approach

- **Knitro** requires the computation of the gradient and **Hessian of each constraint at each iteration**
- A major challenge. It gets in the way of modeling. Very error prone. ACOPF is especially challenging. It should be made automatic
- We used **AMPL** as an intermediary between our Python code and Knitro
- AMPL is **very good** at this !!! Fast (and correct) even on large GO2 models
- But: AMPL is old. It uses **a file system interface** with solvers. Files written and read as inputs and outputs. Can be nontrivial to debug models
- The file system feature interfered with MPI and the testing platform. Richard had to invest time to fix this.
- **It would be outstanding if the AMPL team addressed the file system feature. It belongs in another era**
- Nevertheless, we are very thankful for their help. AMPL proved **invaluable**

# Comments on problem

- On the largest cases, **Knitro** can require multiple minutes to run on a problem "from scratch"

  Noteworthy: includes integer (binary) variables only

  We model integer variables as (appropriate) sums over binary variables

  Adjusted formulation: sometimes we are strict on (not) allowing slacks. Sometimes we penalize slacks linearly

- Thousands of contingencies

- A "proper" security-constrained algorithm should make multiple passes over the contingencies, and re-run the base case in full

- We deemed such an algorithm *infeasible* for our algorithmic setup under the GO2 rules. Not nearly enough time

- So, what to do:
  1. **One-way** algorithm: run the base case, and then run each contingency, repairing the solution as best as possible
  2. But each "run" involves multiple passes

- Why? We expected **realistic** problems in the following regard:
  1. **The prior solution, in general, should be "not too bad",**
  2. **We expected the power system to be flexible**
  3. **We expected that in general the contingencies should be manageable**

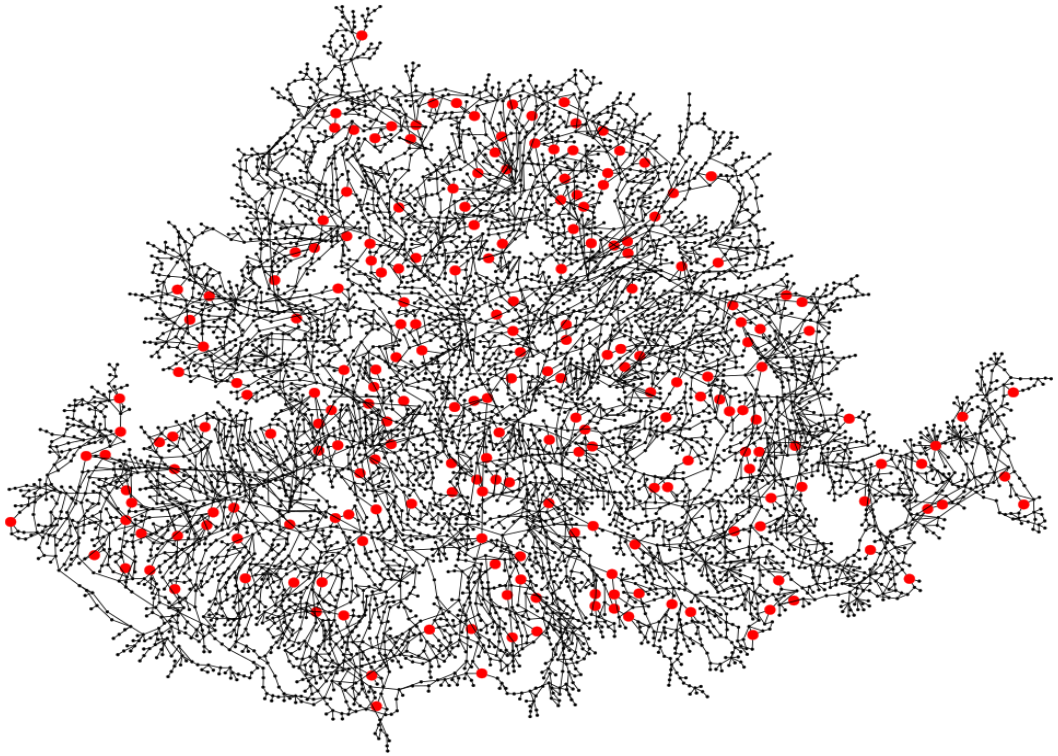
# We like a good challenge: let's be scrappy

- A multiple attempt approach.  Let's consider **the base case**
- First, try the prior solution and keep it as a candidate
- Run Knitro on the relaxation, and then try to round (another candidate)
- Run Knitro on the relaxation, then fix many non-integer variables, then re-run full problem
- Let's cheat:
  - Observe where (which buses) the prior solution is infeasible beyond a threshold
  - Fix integer variables **elsewhere**
  - **Run problem** with the integer variables fixed this way
- Why?

# N06100 scenario 115

6476 buses, 3371 loads, 406 generators, 5337 lines, 3086 transformers, 2467 contingencies

- About 100,000 variables and constraints
- Picture shows buses where prior solution has infeasibility greater than 1e-3:  about 200



obj=712281.64
total_bus_cost 1.85847217e-02
total_load_benefit 1.26257799e+06
total_gen_cost 5.50296330e+05
total_line_cost 0.00000000e+00
total_xfmr_cost 0.00000000e+00

(Knitro feaserror 2.481e-09)

169.05 sec 450 iterations

# How about the contingencies?

- Consider a given contingency
- The **base case** solution is a candidate: let's keep it
- Even if bad (which it often is) the base case solution is bad in **at most two buses**
- We suspect that patching the base case solution primarily involves voltage adjustments
- We view this as primarily a reactive power issue
- An old power engineer's saying: *reactive power does not travel far*

**Solution**

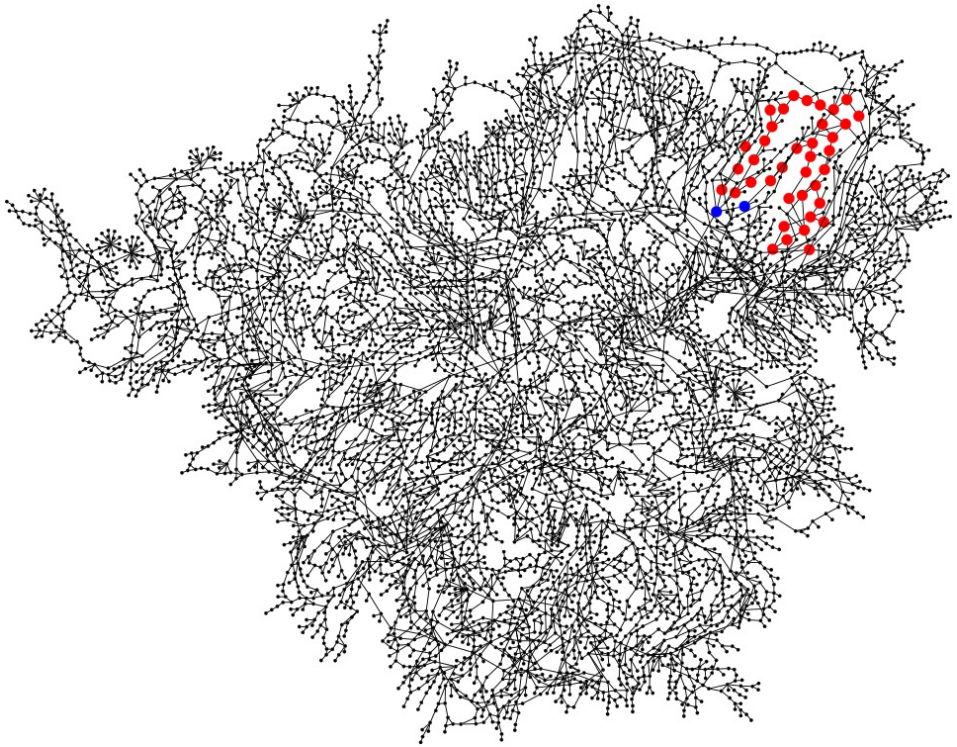
1. Let **S** be the set of buses where the base case solution is infeasible
2. Let **N** be the set of buses that are close to **S** using an ***appropriate metric***
3. **Fix** all binary variables **outside** of **N**. Run Knitro on resulting problem

4. What is the "appropriate metric". Length of a branch **<f, t>** = reactance
5. What is "close": take the nearest K buses. K = 5, 10, 40, 100 were tried by us. Make sure N includes at least one generator

# N06100 scenario 115

6476 buses, 3371 loads, 406 generators, 5337 lines, 3086 transformers, 2467 contingencies

Contingency 3 (line down)



- **Base case** obj: **712281.64**

- **Heuristic** solution:

total_bus_cost 1.65362151e-02
total_load_benefit 1.25961077e+06
total_gen_cost 5.44767392e+05
total_line_cost 0.00000000e+00
total_xfmr_cost 0.00000000e+00

**objective 714843.36**

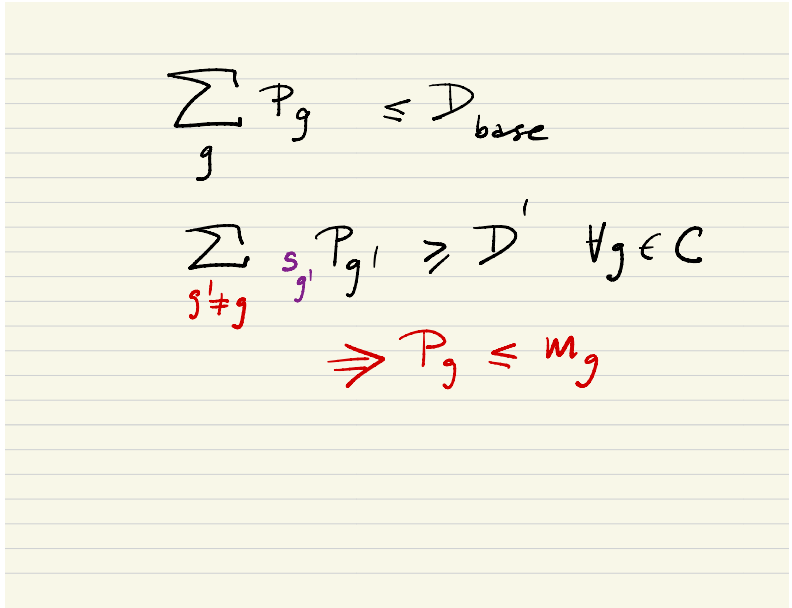
31.30 seconds 227 iterations
+ 6.8 seconds + 27 iterations (rounding)

# Thoughts about the future

- **Implementation in C?** *Assuming the input format and evaluation does not change.*

- Thinking about a GPU-based prox-like algorithm for nonconvex optimization

- We were implementing a "survivable network" step that did not make it into the final round due to our lack of time. *(1 out of N system)*
  - Applies to generator-out contingencies
  - No generator involved in a contingency should output "too much" in the base case solution
  - The "too much" involves a computation that in principle is another nonconvex optimization problem
  - Useful?



# Thoughts about the future

- We look forward to the next challenge

- Please, please, please let it span **three** summers.

Many of us

~~have a day job~~ ~~have a life~~ ~~teach~~

~~have other time consuming ARPA-E contracts~~

have a lot to do beyond this nice competition